Package 'tfCox'

May 8, 2026

Type Package

Title Fits Piecewise Polynomial with Data-Adaptive Knots in Cox Model

Version 0.1.0

Date 2019-07-29

Description In Cox's proportional hazard model, covariates are modeled as linear function and may not be flexible. This package implements additive trend filtering Cox proportional hazards model as proposed in Jiacheng Wu & Daniela Witten (2019) ``Flexible and Interpretable Models for Survival Data'', Journal of Computational and Graphical Statistics, <DOI:10.1080/10618600.2019.1592758>. The fitted functions are piecewise polynomial with adaptively chosen knots.

License GPL (>= 2)

Imports Rcpp (>= 0.12.14), survival, stats

LinkingTo Rcpp

NeedsCompilation yes

Author Jiacheng Wu [aut, cre],
Daniela Witten [aut],
Taylor Arnold [ctb],
Veeranjaneyulu Sadhanala [ctb],
Ryan Tibshirani [ctb]

Maintainer Jiacheng Wu <wujiacheng1992@gmail.com>

Repository CRAN

Date/Publication 2019-08-01 11:50:03 UTC

Contents

`tfCox-package` *Fit the Additive Trend Filtering Cox Model*

Description

This package is called tfCox or trend filtering for Cox model, which is proposed in Jiacheng Wu & Daniela Witten (2019) Flexible and Interpretable Models for Survival Data, Journal of Computational and Graphical Statistics, DOI: 10.1080/10618600.2019.1592758. It provides an approach to fit additive Cox model in which each component function is estimated to be piecewise polynomial with adaptively-chosen knots.

Function `tfCox` fits the trend filtering Cox model for a range of tuning parameters. Function `cv_tfCox` returns the optimal tuning parameter selected by K-fold cross validation.

Details

Package:	tfCox
Type:	Package
Version:	0.1.0
Date:	2019-05-20
License:	GPL (>= 2)

The package includes the following functions: `tfCox`, `cv_tfCox`, `plot.tfCox`, `plot.cv_tfCox`, `predict.tfCox`, `summary.tfCox`, `summary.cv_tfCox`, `sim_dat`, `plot.sim_dat`.

Author(s)

Jiacheng Wu Maintainer: Jiacheng Wu <wujiacheng1992@gmail.com>

References

Jiacheng Wu & Daniela Witten (2019) Flexible and Interpretable Models for Survival Data, Journal of Computational and Graphical Statistics, DOI: 10.1080/10618600.2019.1592758

cv_tfCox *Fit Trend Filtering Cox model and Choose Tuning Parameter via K-Fold Cross-Validation*

Description

Fit additive trend filtering Cox model where each component function is estimated to be piecewise constant or polynomial. Tuning parameter is selected via k-fold cross-validation.

Usage

```
cv_tfCox(dat, ord=0, alpha=1, discrete=NULL, lambda.seq=NULL,
lambda.min.ratio=0.01, n.lambda=30, n.fold=5, seed=NULL, tol=1e-6,
niter=1000, stepSize=25,  backtracking=0)
```

Arguments

dat	A list that contains time, status and X. time is failure or censoring time, status is censoring indicator, and X is n x p matrix and may have p > n.
ord	The polynomial order of the trend filtering fit; a non-negative interger (ord>= 3 is not recommended). For instance, ord=0 will produce piewise constant fit, ord=1 will produce piewise linear fit, and ord=2 will produce piewise quadratic fit.
alpha	The trade-off between trend filtering penalty and group lasso penalty. It must be in [0,1]. alpha=1 corresponds to the case with only trend filtering penalty to produce piecewise polynomial, and alpha=0 corresponds to the case with only group lasso penalty to produce sparsity of the functions. alpha between 0 and 1 is the tradeoff between the strength of these two penalties. For p < n, we suggest using 1.
discrete	A vector of covariate/feature indice that are discrete. Discrete covariates are not penalized in the model. Default NULL means that none of the covariates are discrete thus all covariates will be penalized in the model.
lambda.seq	The sequence of positive lambda values to consider. The default is NULL, which calculates lambda.seq using lambda.min.ratio and n.lambda. If lambda.seq is provided, it will override the default. lambda.seq should be a decreasing positive sequence of values since cv_tfCox replies on warm starts to speed up the computation.
lambda.min.ratio	Smallest value for lambda.seq, as a fraction of the maximum lambda value, which is the smallest value such that the penalty term is zero. The default is 0.01.
n.lambda	The number of lambda values to consider. Default is 30.
n.fold	The number of folds for cross-validation of lambda. The default is 5.
seed	An optional number used with set.seed().

tol	Convergence criterion for estimates.
niter	Maximum number of iterations.
stepSize	Iniitial step size. Default is 25.
backtracking	Whether backtracking should be used 1 (TRUE) or 0 (FALSE). Default is 0 (FALSE).

Details

Note that cv_tfCox does not cross-validate over alpha, and alpha should be provided. However, if the user would like to cross-validate over alpha, then cv_tfCox should be called multiple times for different values of alpha and the same seed. This ensures that the cross-validation folds (fold) remain the same for the different values of alpha. See the example below for details.

Value

An object with S3 class "cv_tfCox".

best.lambda	Optional lambda value chosen by cross-dalidation.
lambda.seq	lambda sequence considered.
mean.cv.error	vector of average cross validation error with the same length as lambda.seq

Author(s)

Jiacheng Wu

References

Jiacheng Wu & Daniela Witten (2019) Flexible and Interpretable Models for Survival Data, Journal of Computational and Graphical Statistics, DOI: 10.1080/10618600.2019.1592758

See Also

summary.cv_tfCox, plot.cv_tfCox, tfCox

Examples

```
#generate data
set.seed(123)
dat = sim_dat(n=100, zerof=0, scenario=1)

#fit piecewise constant functions
#cross-validation to choose the tuning parameter lambda with fixed alpha=1
cv = cv_tfCox(dat, ord=0, alpha=1, n.fold=2, seed=123)
plot(cv, showSE=TRUE)
```

negloglik *Calculate the negative log likelihood from Cox model.*

Description

Calculate the negative log likelihood from Cox model from the estimated coefficient matrix theta.

Usage

```
negloglik(dat, theta)
```

Arguments

dat	A list that contains time, status and X. time is failure or censoring time, status is censoring indicator, and X is n x p matrix and may have p > n.
theta	An n x p matrix of coefficients corresponding to covariates X.

Author(s)

Jiacheng Wu

References

Jiacheng Wu & Daniela Witten (2019) Flexible and Interpretable Models for Survival Data, Journal of Computational and Graphical Statistics, DOI: 10.1080/10618600.2019.1592758

See Also

predict_best_lambda, tfCox_choose_lambda

Examples

```
#generate training and testing data
dat = sim_dat(n=100, zerof=0, scenario=1)
test_dat = sim_dat(n=100, zerof=0, scenario=1)

#choose the optimal tuning parameter
cv = tfCox_choose_lambda(dat, test_dat, ord=0, alpha=1)
plot(cv$lam_seq, cv$loss)

#optimal tuning parameter
cv$best_lambda

#predict the coefficients of testing covariates from the optimal tuning parameter
#from tfCox_choose_lambda object.
theta_hat = predict_best_lambda(cv, test_dat$X)

#calculate the loss in the testing data based on the estimated coefficients theta
negloglik(test_dat, theta_hat)
```

plot.cv_tfCox *Plots Cross-Validation Curve for Object of Class "cv_tfCox"*

Description

This function plots the cross-validation curve for models fitted by a range of tuning parameter lambda using cv_tfCox. The cross-validation error with +/-1 standard error is plotted for each value of lambda. The dotted vertical line indicates the chosen lambda corresponding to the minimum cross-validation error.

Usage

```
## S3 method for class 'cv_tfCox'
plot(x, showSE=F, ...)
```

Arguments

x	an object of class "cv_tfCox"
showSE	a logical (TRUE or FALSE) for whether the standard errors of the curve should be plotted
...	additional arguments to be passed. These are ignored in this function.

Author(s)

Jiacheng Wu

See Also

cv_tfCox

Examples

```
#generate data
set.seed(123)
dat = sim_dat(n=100, zerof=0, scenario=1)

#fit piecewise constant functions
#cross-validation to choose the tuning parameter lambda with fixed alpha=1
cv = cv_tfCox(dat, ord=0, alpha=1, n.fold=2, seed=123)
plot(cv, showSE=TRUE)
```

plot.sim_dat *Plot the true covariate effects*

Description

This function plots the functional form of covariate effects in four simulation scenarios.

Usage

```
## S3 method for class 'sim_dat'
plot(x, which.predictor = NULL, n.plot = 4, ...)
```

Arguments

x	an object of class "sim_dat"
which.predictor	a vector of predictor index that indicates which predictor function to plot. The vector should have integer values from 1 to p where p is the number of predictors.
n.plot	the number of predictors to be plotted (default is 4). If which.predictor is supplied, this argument is ignored.
...	additional arguments to be passed. These are ignored in this function.

Author(s)

Jiacheng Wu

See Also

sim_dat

Examples

```
#generate data
set.seed(123)
dat = sim_dat(n=100, zerof=0, scenario=1)
#plot X versus the true theta
plot.sim_dat(dat)
```

plot.tfCox *Plot Fitted Functions from Class "tfCox"*

Description

This function plots the fitted functions from a model estimated by tfCox.

Usage

```
## S3 method for class 'tfCox'
plot(x, which.lambda=1, which.predictor = NULL, n.plot = 4, ...)
```

Arguments

x	an object of class "tfCox"
which.lambda	the index for the model of interest to be plotted. which.lambda corresponds to the model fit in lambda.seq and should be integer between 1 to length(fit$lambda.seq). In other words, the fit from fit$theta.list[[which.lambda]] will be plotted.
which.predictor	a vector of predictor index that indicates which predictor function to plot. The vector should have integer values from 1 to p where p is the number of predictors.
n.plot	the number of predictors to be plotted (default is 4). Note that only those non-zero estimated functions will be plotted. If which.predictor is supplied, this argument is ignored.
...	additional arguments to be passed. These are ignored in this function.

Author(s)

Jiacheng Wu

See Also

tfCox

Examples

```
#generate data
set.seed(123)
dat = sim_dat(n=100, zerof=0, scenario=1)

fit = tfCox(dat, ord=0, alpha=1, lambda.seq=0.04)
plot(fit, n.plot=4)
```

predict.tfCox *Predict for a New Covariate Matrix and fit from* tfCox

Description

This function makes predictions from a specified covariate matrix for a fit of the class "tfCox".

Usage

```
## S3 method for class 'tfCox'
predict(object, newX, which.lambda=1, ...)
```

Arguments

object	an object of the class "tfCox"
newX	a n x p covariate matrix
which.lambda	the index for the model of interest to be plotted. which.lambda corresponds to the model fit in lambda.seq and should be integer between 1 to length(fit$lambda.seq). In other words, the fit from fit$theta.list[[which.lambda]] will be plotted.
...	additional arguments to be passed. These are ignored in this function.

Details

Prediction for the new data point is implemented by constant or linear interpolation. 0th order trend filtering will have constant interpolation, and 1th or higher order trend filtering will have linear interpolation.

Value

A n x p matrix containing the fitted theta values.

Author(s)

Jiacheng Wu

See Also

tfCox

predict_best_lambda *Predict from the optimal lambda from tfCox_choose_lambda*

Description

Estimate the corresponding theta values from the optimal tuning parameter obtained by tfCox_choose_lambda.

Usage

```
predict_best_lambda(cv, newX)
```

Arguments

cv	An object from tfCox_choose_lambda.
newX	The new covariate values.

Value

Estimated theta values.

Author(s)

Jiacheng Wu

References

Jiacheng Wu & Daniela Witten (2019) Flexible and Interpretable Models for Survival Data, Journal of Computational and Graphical Statistics, DOI: 10.1080/10618600.2019.1592758

See Also

tfCox_choose_lambda, negloglik

Examples

```
#generate training and testing data
dat = sim_dat(n=100, zerof=0, scenario=1)
test_dat = sim_dat(n=100, zerof=0, scenario=1)

#choose the optimal tuning parameter
cv = tfCox_choose_lambda(dat, test_dat, ord=0, alpha=1)
plot(cv$lam_seq, cv$loss)

#optimal tuning parameter
cv$best_lambda

#Estimate the theta values of testing covariates from the optimal tuning parameter
#from tfCox_choose_lambda object.
theta_hat = predict_best_lambda(cv, test_dat$X)
```

sim_dat *Simulate Data from a Variety of Functional Scenarios*

Description

This function generates survival data according to the simulation scenarios considered in Section 4 of Wu, J., and Witten, D. (2019) Flexible and interpretable models for survival data. Cox model has the form

$$\lambda(t|x) = \lambda_0(t)exp(\sum_{j=1}^{p} f_j(x))$$

. Failure time is generated by Weibull distribution with baseline hazard

$$\lambda_0(t) = scale * shape * t^{shape-1}$$

. In the paper, however, failure time is generated by a simplied weibull distribution: exponential(1) baseline hazard corresponding to `shape=1` and `scale=1`. Censoring time is generated independently by exponential distribution with intensity `censoring.rate`. Thus the observed time is the minimum of failure time and censoring time. Each scenario has four covariates that have some non-linear association with the outcome. There is the option to also generate a user-specified number of covariates that have no association with the outcome.

Usage

```
sim_dat(n, zerof=0, scenario=1, scale=1, shape=1, censoring.rate=0.01, n.discrete=0)
```

Arguments

`n`	number of observations.
`scenario`	Simulation scenario. Options are 1, 2, 3, 4. Scenario 1 corresponds to piecewise constant functions, scenario 2 corresponds to smooth functions, scenario 3 corresponds to piecewise linear functions, and scenario 4 corresponds to functions that have varying degrees of smoothness. Each scenario has four covariates that have some non-linear association with the outcome.
`zerof`	Number of additional covariates that have no association with the outcome. The total number of covariates is `4+zerof`.
`scale`	scale parameter as in `rweibull`
`shape`	shape parameter as in `rweibull`
`censoring.rate`	censoring intensity. Censoring time is generated by exponential distribution with intensity `censoring.rate`.
`n.discrete`	The number of binary covariates and default is zero binary covariate.

Value

`time`	failure or censoring time whichever comes first.
`status`	censoring indicator. 1 denotes censoring and 0 denotes failure.
`X`	n x p covariate matrix.
`true_theta`	n x p matrix.

Author(s)

Jiacheng Wu

References

Jiacheng Wu & Daniela Witten (2019) Flexible and Interpretable Models for Survival Data, Journal of Computational and Graphical Statistics, DOI: 10.1080/10618600.2019.1592758

See Also

plot.sim_dat

Examples

```
#generate data
set.seed(123)
dat = sim_dat(n=100, zerof=0, scenario=1)
#plot X versus the true theta
plot.sim_dat(dat)
```

---

summary.cv_tfCox *Summarize* cv_tfCox *object*

Description

This function summarizes cv_tfCox object and identifies the tuning parameter chosen by cross-validation.

Usage

```
## S3 method for class 'cv_tfCox'
summary(object, ...)
```

Arguments

object	an object of class "cv_tfCox"
...	additional arguments to be passed. These are ignored in this function.

Author(s)

Jiacheng Wu

See Also

cv_tfCox, plot.cv_tfCox

Examples

```
#generate data
set.seed(1234)
dat = sim_dat(n=100, zerof=0, scenario=1)

#cross-validation to choose the tuning parameter lambda with fixed alpha=1
cv = cv_tfCox(dat, ord=0, alpha=1, n.fold=2)
#summarize the cross-validation
summary(cv)
#plot the cross-validation curve
plot(cv)
```

`summary.tfCox` *Summarize* `tfCox` *object*

Description

This function summarizes `tfCox` object

Usage

```
## S3 method for class 'tfCox'
summary(object, ...)
```

Arguments

`object`	an object of class "tfCox"
`...`	additional arguments to be passed. These are ignored in this function.

Details

Summarize the fit by the number of knots and percent sparsity achieved. Percent sparsity is the percentage of features estimated to have no relationship with the outcome.

Author(s)

Jiacheng Wu

See Also

`tfCox`, `plot.tfCox`

Examples

```
#generate data
set.seed(1234)
dat = sim_dat(n=100, zerof=0, scenario=1)

#fit piecewise constant for alpha=1 and a range of lambda
fit = tfCox(dat, ord=0, alpha=1)

#summarize the fit by the number of knots and percent sparsity achieved.
#Percent sparsity is the percentage of features estimated to have
#no relationship with outcome
summary(fit)
```

`tfCox` *Fit the additive trend filtering Cox model with a range of tuning parameters*

Description

Fit additive trend filtering Cox model where each component function is estimated to be piecewise constant or polynomial.

Usage

```
tfCox(dat, ord=0, alpha=1, lambda.seq=NULL, discrete=NULL, n.lambda=30,
lambda.min.ratio = 0.01, tol=1e-6, niter=1000, stepSize=25, backtracking=0)
```

Arguments

`dat`	A list that contains `time`, `status` and `X`. `time` is failure or censoring time, `status` is failure indicator with 1 indicating failure and 0 indicating censoring, and `X` is n x p design matrix and may have p > n. Missing data are not allowed in `time`, `status` and `X`. `X` should be numeric.
`ord`	The polynomial order of the trend filtering fit; a non-negative interger (`ord>= 3` is not recommended). For instance, `ord=0` will produce piewise constant fit, `ord=1` will produce piewise linear fit, and `ord=2` will produce piewise quadratic fit.
`alpha`	The trade-off between trend filtering penalty and group lasso penalty. It must be in [0,1]. `alpha=1` corresponds to the case with only trend filtering penalty to produce piecewise polynomial, and `alpha=0` corresponds to the case with only group lasso penalty to produce sparsity of the functions. `alpha` between 0 and 1 is the tradeoff between the strength of these two penalties. For p < n, we suggest using 1.
`lambda.seq`	A vector of non-negative tuning parameters. If provided, `lambda.seq` should be a decreasing sequence of values since `tfCox` uses warm starts for speed. If `lambda.seq=NULL`, the default will calculate `lambda.seq` using `lambda.min.ratio` and `n.lambda`.

discrete	A vector of covariate/feature indice that are discrete. Discrete covariates are not penalized in the model. Default NULL means that none of the covariates are discrete thus all covariates will be penalized in the model.
n.lambda	The number of lambda values to consider and the default is 30.
lambda.min.ratio	Smallest value for lambda.seq, as a fraction of the maximum lambda value, which is the smallest value such that the penalty term is zero. The default is 0.01.
tol	Convergence criterion for estimates.
niter	Maximum number of iterations.
stepSize	Initial step size. Default is 25.
backtracking	Whether backtracking should be used 1 (TRUE) or 0 (FALSE). Default is 0 (FALSE).

Details

The optimization problem has the form

$$l(\theta) + \alpha\lambda \sum_{j=1}^{p} |D_j P_j \theta_j|_1 + (1-\alpha)\lambda \sum_{j=1}^{p} |\theta_j|_2$$

where l is the loss function defined as the negative log partial likelihood divided by n, and α provides a trade-off between trend filtering penalty and group lasso penalty. Covariate matrix X is not standardized before solving the optimization problem.

Value

An object with S3 class "tfCox".

ord	the polynomial order of the trend filtering fit. Specified by user (or default).
alpha	as specified by user (or default).
lambda.seq	vector of lambda values considered.
theta.list	list of estimated theta matrices of dimension n x p. Each component in the list corresponds to the fit from lambda.seq.
num.knots	vector of number of knots of the estimated theta. Each component corresponds to the fit from lambda.seq.
num.nonsparse	vector of proportion of non-sparse/non-zero covariates/features. Each component corresponds to the fit from lambda.seq.
dat	as specified by user.

Author(s)

Jiacheng Wu

References

Jiacheng Wu & Daniela Witten (2019) Flexible and Interpretable Models for Survival Data, Journal of Computational and Graphical Statistics, DOI: 10.1080/10618600.2019.1592758

See Also

summary.tfCox, predict.tfCox, plot.tfCox, cv_tfCox

Examples

```
#####################################################################
#constant trend filtering (fused lasso) with adaptively chosen knots
#generate data from simulation scenario 1 with piecewise constant functions
set.seed(1234)
dat = sim_dat(n=100, zerof=0, scenario=1)

#fit piecewise constant for alpha=1 and a range of lambda
fit = tfCox(dat, ord=0, alpha=1)
summary(fit)
#plot the fit of lambda index 15 and the first predictor
plot(fit, which.lambda=15, which.predictor=1)

#cross-validation to choose the tuning parameter lambda with fixed alpha=1
cv = cv_tfCox(dat, ord=0, alpha=1, n.fold=2)
summary(cv)
cv$best.lambda
#plot the cross-validation curve
plot(cv)

#fit the model with the best tuning parameter chosen by cross-validation
one.fit = tfCox(dat, ord=0, alpha=1, lambda.seq=cv$best.lambda)
#predict theta from the fitted tfCox object
theta_hat = predict(one.fit, newX=dat$X, which.lambda=1)

#plot the fitted theta_hat (line) with the true theta (dot)
for (i in 1:4) {
  ordi = order(dat$X[,i])
  plot(dat$X[ordi,i], dat$true_theta[ordi,i],
    xlab=paste("predictor",i), ylab="theta" )
  lines(dat$X[ordi,i], theta_hat[ordi,i], type="s")
}


#####################################################################
#linear trend filtering with adaptively chosen knots
#generate data from simulation scenario 3 with piecewise linear functions
set.seed(1234)
dat = sim_dat(n=100, zerof=0, scenario=3)

#fit piecewise constant for alpha=1 and a range of lambda
fit = tfCox(dat, ord=1, alpha=1)
summary(fit)
```

```
#plot the fit of lambda index 15 and the first predictor
plot(fit, which.lambda=15, which.predictor=1)

#cross-validation to choose the tuning parameter lambda with fixed alpha=1
cv = cv_tfCox(dat, ord=1, alpha=1, n.fold=2)
summary(cv)
#plot the cross-validation curve
plot(cv)

#fit the model with the best tuning parameter chosen by cross-validation
one.fit = tfCox(dat, ord=1, alpha=1, lambda.seq=cv$best.lambda)
#predict theta from the fitted tfCox object
theta_hat = predict(one.fit, newX=dat$X, which.lambda=1)

#plot the fitted theta_hat (line) with the true theta (dot)
for (i in 1:4) {
  ordi = order(dat$X[,i])
  plot(dat$X[ordi,i], dat$true_theta[ordi,i],
       xlab=paste("predictor",i), ylab="theta" )
  lines(dat$X[ordi,i], theta_hat[ordi,i], type="l")
}
```

`tfCox_choose_lambda` *Choose the tuning parameter lambda using training and testing dataset*

Description

Fit additive trend filtering Cox model where each component function is estimated to be piecewise constant or polynomial. Tuning parameter is selected via training and testing dataset described in Wu and Witten (2019). Training data is used to build the model, and testing data is used for selecting tuning parameter based on log likelihood. It is a convenience function to replicate the simulation results in Wu and Witten (2019).

Usage

```
tfCox_choose_lambda(dat, test_dat, ord = 0, alpha = 1, discrete = NULL,
lam_seq = NULL, nlambda = 30, c = NULL, tol = 1e-06, niter=1000,
stepSize=25, backtracking=0)
```

Arguments

`dat`	A list that contains `time`, `status` and `X`. `time` is failure or censoring time, `status` is censoring indicator, and `X` is n x p matrix and may have p > n. This is the training data that will be used for estimation for a given tuning parameter lambda.

test_dat	Same list frame as before. This is the testing data that will be used for selecting tuning parameter based on the log likelihood fit.
ord	The polynomial order of the trend filtering fit; a non-negative interger (ord>= 3 is not recommended). For instance, ord=0 will produce piewise constant fit, ord=1 will produce piewise linear fit, and ord=2 will produce piewise quadratic fit.
alpha	The trade-off between trend filtering penalty and group lasso penalty. It must be in [0,1]. alpha=1 corresponds to the case with only trend filtering penalty to produce piecewise polynomial, and alpha=0 corresponds to the case with only group lasso penalty to produce sparsity of the functions. alpha between 0 and 1 is the tradeoff between the strength of these two penalties. For p < n, we suggest using 1.
discrete	A vector of covariate/feature indice that are discrete. Discrete covariates are not penalized in the model. Default NULL means that none of the covariates are discrete thus all covariates will be penalized in the model.
lam_seq	The sequence of positive lambda values to consider. The default is NULL, which calculates lambda.seq using lambda.min.ratio and n.lambda. If lambda.seq is provided, it will override the default. lambda.seq should be a decreasing positive sequence of values since cv_tfCox replies on warm starts to speed up the computation.
nlambda	The number of lambda values to consider. Default is 30.
c	Smallest value for lam_seq, as a fraction of the maximum lambda value, which is the smallest value such that the penalty term is zero. The default is NULL.
tol	Convergence criterion for estimates.
niter	Maximum number of iterations.
stepSize	Iniitial step size. Default is 25.
backtracking	Whether backtracking should be used 1 (TRUE) or 0 (FALSE). Default is 0 (FALSE).

Value

lam_seq	Lambda sequence considered.
loss	Loss based on the testing data with the same length as lambda.seq
knots	Number of knots from the training data with the same length as lambda.seq
paramfit	Mean square error between the estimated and true theta for the testing data.
best_lambda	The lambda that achieves the minimum loss for testing data.

Author(s)

Jiacheng Wu

References

Jiacheng Wu & Daniela Witten (2019) Flexible and Interpretable Models for Survival Data, Journal of Computational and Graphical Statistics, DOI: 10.1080/10618600.2019.1592758

See Also

predict_best_lambda, negloglik

Examples

```
#generate training and testing data
dat = sim_dat(n=100, zerof=0, scenario=1)
test_dat = sim_dat(n=100, zerof=0, scenario=1)

#choose the optimal tuning parameter
cv = tfCox_choose_lambda(dat, test_dat, ord=0, alpha=1)
plot(cv$lam_seq, cv$loss)

#optimal tuning parameter
cv$best_lambda

#predict the coefficients of testing covariates from the optimal tuning parameter
#from tfCox_choose_lambda object.
theta_hat = predict_best_lambda(cv, test_dat$X)

#calculate the loss in the testing data based on the estimated coefficients theta
negloglik(test_dat, theta_hat)
```

Index